# Cost Graphs for Concurrent Calculi

AUTHOR(S):

Kubo, Makoto

CITATION:

Kubo, Makoto. Cost Graphs for Concurrent Calculi. 数理解析研究所講究録 1995, 902: 64-79

ISSUE DATE:

1995-03

URL:

http://hdl.handle.net/2433/59388

数理解析研究所講究録
902巻 1995年 64-79



# Cost Graphs for Concurrent Calculi

久保 誠* Makoto Kubo

Department of Computer Science, Keio University
3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223, Japan

**Abstract**

We present a theory of cost of concurrent programs, which is an important measure to evaluate their quality. The basic idea is to assign each program a set of graphs which give derivation of a program argumented with cost information, then order two programs by comparing these two sets. According to the ways of comparing these sets, various orderings which capture specific aspects of efficiency of concurrent programs arise. We show that the efficiency preorder presented by Arun-Kumar and Hennessy, is closely related with one of these orders, thus showing our theory is a conservative extension of their construction.

## 1 Introduction

One essential quality of a program is its efficiency or *cost* of computation. If two programs can achieve the same thing, we always prefer the one which runs faster or at least the one which does the work "in time"[1]. In fact this is one of the most important concerns in theoretical computer science, where the formal transformation methods for program optimization is a long-lasting and still active field of study. Now for any such methodology, we should have a basic notion of what we mean by cost of programs. While this may not be easy in the real applications, some clean idea in the abstract setting should surely help, giving the basic criteria and a reference framework for further ramifications, offering a formal way of evaluating the quality of software components.

In the sequential programs, one clean answer exists, which says an abstract cost of a program can be measured in terms of the number of (elementary) computation steps needed for it to reach the final answer. This is the basis of the theory of computational complexity and Levy's notion of *optimal reduction*[7]. This idea comes from the fact that sequential computation may not make sense if a program has nontermination. Yet when we turn to the concurrency world, where the computation proceeds by processes or objects which communicate with each other, this simple answer cannot be adapted easily, since the underlying semantic framework is quite different. Specifically:

*kubo@mt.cs.keio.ac.jp

[1]The present paper does not consider so-called *space-cost* of programs.

- In concurrency, it is often the case that the intermediate action sequence matters. In fact, in many cases concurrent programs are made not to compute the single answer, but to *interact* with the outside world, e.g. operating systems or banking systems.

- Relatedly, concurrent computation may make sense even if a program has nontermination. In fact the banking system is not expected to stop at all!

- In addition, concurrent computation is essentially non-deterministic due to the existence of interference (or competition). So there can be many possible nondeterministic branching executions of the same program.

Therefore we should consider not only the number of computation steps but also the properties such as nondeterminism, nontermination, intermediate steps, etc. This makes it difficult to have a clean idea about what cost means in concurrency, even in the abstract setting. Considering the importance of cost in software evaluation, we think that a formal framework for concurrent computing in this regard is absolutely needed since a program becomes meaningful only when they are implemented to run in the real machine and architecture.

Under the above background, this paper introduces an elementary theory of cost or efficiency for concurrent programs, based on the structure called *cost graphs*. Just as the theoretical study of sequential programs often uses $\lambda$-calculi as its theoretical basis, we use process calculi (e.g. CSP[4], CCS[9], $\pi$-calculus [11] and many others) as our basic formal theories of concurrent computation. A cost graph is a graph which traces the actions a process may take considering their cost as well as branching structures. Due to a nondeterministic behaviour, a program can be given a *set of* cost graphs, which are essentially isomorphic to each other, if we forget the cost information. Cost of programs is evaluated in terms of the whole set of such graphs. While we assume that the cost of computation is the number of transition steps as in $\lambda$-calculi, the ways of measuring and comparing cost are quite different from those in sequential programs.

There exists earlier work [1] by Arun-Kumar and Hennessy, which considers the cost of computation in the concurrency setting. [1] defines the order over CCS terms by the number of $\tau$ transitions of the labelled transition system of CCS, taking the nondeterministic behaviour into consideration. The definition of the ordering is inductive just like bisimulation, offering a convenient proof methods. Assuming a prior knowledge of CCS [9], let us give a definition of *efficiency prebisimulation* over CCS terms. It is a binary relation $R \subseteq P \times P$ such that, for every $< P, Q > \in R$,

1. $P \xrightarrow{s} P' \;\Rightarrow\; \exists s'\, s \leq s'\, \exists Q'\, Q \xrightarrow{s'} Q' \wedge\; < P', Q' > \in R$

2. $Q \xrightarrow{s'} Q' \;\Rightarrow\; \exists s\, s \leq s'\, \exists P'\, P \xrightarrow{s} P' \wedge\; < P', Q' > \in R$

where $s, s'$ is the sequence of labels and $s \leq s'$ is given as $\alpha.\tau.\beta.0 \leq \alpha.\beta.0$ etc. The theory offers, as far as we know, the first coherent notion of concurrency cost in the line of behavioural equivalence over processes. Note that the ordering relation is essentially convex, thus expressing a *relative* notion of cost between two terms. Thus even if $P$ is ordered as superior to $Q$, $P$ may not run faster than $Q$ if we have a bad luck.

Regarding Arun-Kumar-Hennessy construction as an important precursor, we note that there might be several features which we may need for a cost theory of computation but which cannot be covered by the above notion. For example, there seems to be no obvious way of defining "absolute" comparison (one program runs absolutely faster than another, etc.). Moreover we may face a subtle situation where we may want a different measure of evaluating cost. Let us think of two terms, $a.P|\Omega$ and $\tau.a.P$, where $\Omega \stackrel{\text{def}}{=} \tau.\Omega$ (note $P|\Omega$ and $P$ is semantically equal w.r.t. bisimilarity). From a viewpoint of cost, the former takes at least one step to reach $P|\Omega$. At the worst case, however, it takes infinite steps. In contrast, it *always* takes two steps for the latter to reach $P$ in any cases. They cannot be compared in efficiency bisimulation in general, but certainly there are situations where we prefer one to the other (e.g. a "speculative" person likes $a.P|\Omega$ while a "steady" person will like $a.\tau.P$). Such worst/base-case analysis seems difficult in the framework of efficiency bisimulation.

Theory of cost graphs are our trial to provide a comprehensive theoretical framework for the study of *cost* in concurrent computation. We take CCS as a base formalism, partly to retain the continuity from the Arun-Kumar-Hennessy construction, partly due to its simplicity and popularity. However the result can be transplanted in any CCS-like formalism for which labelled transition relation as well as $\tau$-action are definable. We show how the set of cost graphs are derivable from a CCS term based on its derivation graph, how we compare two different cost graphs, and, based on the ordering obtained in this way, how various orderings over CCS terms are obtained, each with distinct emphasis on the feature of cost which we may be concerned with. One of such orderings is closely related with efficiency bisimulation

The remainder of the paper is organized as follows. In section 2, we show the construction of derivation of cost graphs from labelled transition systems. Section 3 describe the ordering of the cost graph. And, we explain the examples of comparison of concurrent programs using the presented orders. Section 4 describe the embedding into efficiency preorders. Finally, Section 5 describe the further issues and conclude.

# 2 Cost graph of concurrent computation

## 2.1 CCS and Its labelled transition systems

CCS[8, 9] is a well-known process calculus. The discussions in this paper are based on CCS and its labelled transition system, which we summarize below.

**Definition 2.1** (*the syntax of CCS*)
Let $A$ and $\overline{A}$ be a countable set of names and co-names. Let $L = A \cup \overline{A}$ be a countable set of labels, ranged over by $l, l', \ldots$. Let $Act = L \cup \{\tau\}$ be a countable set of actions, ranged over by $\alpha, \beta, \ldots$. Let $\mathcal{A}$ is a countable set of process variables, ranged over by $X, Y, \ldots$. We define the set of processes $\mathcal{P}$, ranged over by $P, Q, \ldots$, as follows:

$$P = \alpha.P \mid P|Q \mid P\backslash L \mid P + Q \mid X$$

where $X \stackrel{\text{def}}{=} P$.

**Definition 2.2** (*The operational semantics of CCS*)
the transition relation $\xrightarrow{\alpha} \subseteq \mathcal{P} \times Act \times \mathcal{P}$ is satisfying the following conditions.

$$
\begin{array}{ll}
ACT & \alpha.P \xrightarrow{\alpha} P \\
COM & l.P \mid \bar{l}.Q \xrightarrow{\tau} P \mid Q \\
PAR & P \xrightarrow{\alpha} P' \Rightarrow P \mid Q \xrightarrow{\alpha} P' \mid Q \\
RES & P \xrightarrow{\alpha} P' \Rightarrow P\backslash L \xrightarrow{\alpha} P'\backslash L \\
SUM_1 & P \xrightarrow{\alpha} P' \Rightarrow P + Q \xrightarrow{\alpha} P' \\
SUM_2 & Q \xrightarrow{\alpha} Q' \Rightarrow P + Q \xrightarrow{\alpha} Q' \\
REC & P \xrightarrow{\alpha} P' \text{ and } X \stackrel{\text{def}}{=} P \Rightarrow X \xrightarrow{\alpha} P'
\end{array}
$$

**Definition 2.3** (*The labelled transition system of CCS*)
Let $\mathcal{P}$ be a set of processes and $Act$ be a set of actions and $\xrightarrow{l}$ be a transition relation. Triple $T = (P, Act, \xrightarrow{\alpha} \ \alpha \in Act)$ is called a labelled transition system(LTS, in short) of CCS.

**Definition 2.4** (*Sequence of label*)
$s \in Act^*$ is the sequence of actions. $\hat{s} \in L^*$ is the sequence of label.

- $P \xrightarrow{s} Q \stackrel{\text{def}}{=} P \xrightarrow{\tau}{}^* \xrightarrow{\alpha_1} P_2 \xrightarrow{\tau}{}^* \cdots \xrightarrow{\tau}{}^* \xrightarrow{\alpha_n} \xrightarrow{\tau}{}^* Q$ where $s = \alpha_1 \cdots \alpha_n$
- $P \xrightarrow{\hat{s}} Q \stackrel{\text{def}}{=} P \xrightarrow{\tau}{}^* \xrightarrow{l_1} P_2 \xrightarrow{\tau}{}^* \cdots \xrightarrow{\tau}{}^* \xrightarrow{l_n} \xrightarrow{\tau}{}^* Q$ where $\hat{s} = l_1 \cdots l_n$

**Definition 2.5** $reach(P) \stackrel{\text{def}}{=} \{P' \in \mathcal{P} \mid P \in \mathcal{P} \xrightarrow{\alpha}{}^* P'\}$

**Definition 2.6** (*Derivative graph of process P*)
Let $P$ be a processes. An derivative graph of process $P$ is a triple $DG(P) = (N, E)$ satisfying the following conditions.

1. $N$ is a set of *nodes* such that $N \stackrel{\text{def}}{=} reach(P)$
2. $E$ is a set of *labeled directed edge* iff $E \stackrel{\text{def}}{=} \{< P, \alpha, P' > \mid P \xrightarrow{\alpha}{}^* P'\}$.

**Definition 2.7** (*Weak-bisimilarity*)
A binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is a weak bisimulation if $\forall (P, Q) \in R$ and

1. $P \xrightarrow{s} P' \Rightarrow \exists Q' \ Q \xrightarrow{\hat{s}} Q' \wedge (P', Q') \in R$
2. $Q \xrightarrow{s} Q' \Rightarrow \exists P' \ P \xrightarrow{\hat{s}} P' \wedge (P', Q') \in R$ ■

P weak bisimilar Q,written $P \approx Q$, iff $\exists$ weak bisimulation $R, P\mathbf{R}Q$

## 2.2 Construction of cost graph

If we use a labelled transition system to define the semantics of computation, the number of transition steps is naturally used to compare two concurrent programs. So we add the cost data to transition relation as a label, i.e. $P \xrightarrow{l} Q$ becomes $P \xrightarrow{<l,c>} Q$ where $c$ is transition cost. Next, two programs compared with respect to cost should be semantically equal. In this paper, we assume "semantic equality" means weak-bisimilarity, one of the most stable semantic notions for process calculi.

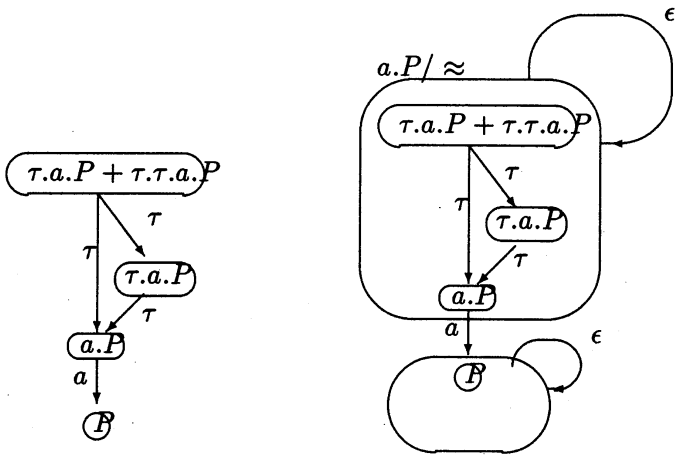


Figure 1: Derivative graph and abstract derivative graphs

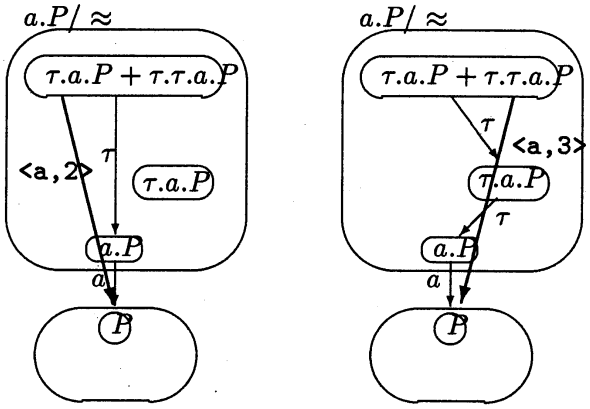


Figure 2: cost graphs from Figure 1

## 2.3 Formal definitions of cost graph

In this section, we define formally the notion of cost graphs. To show the idea, we introduce possibly the most condensed way of presenting transitions modulo $\approx$ as a graph. The structure is later annotated by cost to become a cost graph. Mathematically, this underlying structure is a graph whose nodes are equivalence classes of $\approx$, and whose edges are "semantically atomic" transitions between the equivalence classes, annotated by self-directed edges at each node.

**Notation 2.8**

1. $P \xrightarrow{\tau}_{\approx} P' \stackrel{\text{def}}{=} P \xrightarrow{\tau} P' \wedge P \approx P'$
2. $P \xrightarrow{\tau}^{0,1} P' \stackrel{\text{def}}{=} P' \vee P \xrightarrow{\tau} P'$
3. $P \not\longrightarrow \stackrel{\text{def}}{=}$ for no $\alpha$ *s.t.* $P \xrightarrow{\alpha}$

**Definition 2.9** (*Abstract derivative graph*)
Let $DG(P) = (N, E)$ be the derivative graph of a process $P$, $\approx$ be the weak-bisimilarity on $\mathcal{P}$. Let $L^+ = L \cup \{\epsilon\}$ be the set of abstracted-labels, ranged over by $k, k'$. $DG_{\approx}(P) = (N^P_{\approx}, E_{\approx})$ is called an abstracted-derivative graph iff

1. $N^P_{\approx} \stackrel{\text{def}}{=} N/\approx$ is a set of equivalence classes of $reach(P)$ induced by $\approx$, ranged over by $o, o', \ldots$, with a distinguished element $[P]_{\approx}$.
2. $E_{\approx} \subseteq N_{\approx} \times L^+ \times N_{\approx}$ is a set of labelled directed edges satisfying the following conditions. $< o, k, o' > \in E_{\approx}$ is written $o \xrightarrow{k} o'$.

$$E_{\approx} \stackrel{\text{def}}{=} \begin{cases} \{< o, l, o' > | o, o' \in N_{\approx} \wedge P_1 \xrightarrow{\tau}{}^*_{\approx} \xrightarrow{l} P_2 \wedge P_1 \in o \wedge P_2 \in o'\} \cup \\ \{< o, \epsilon, o' > | o, o' \in N_{\approx} \wedge P_1 \xrightarrow{\tau}{}^*_{\approx} \xrightarrow{\tau}{}^{0,1} P_2 \wedge P_1 \in o \wedge P_2 \in o'\} \end{cases}$$

■

As seen in Proposition 2.10, 2.11, and 2.13, the graph is the faithful and "condensed" representation of transition graphs modulo weak bisimilarity, where no extra edges are present (except adjoined $\epsilon$ edge) but all possible transition paths can be generated from it. We do not know whether this construction has appeared in the literature or not.

The following proposition is important. It shows that the restriction of an abstracted edge in $DG_{\approx}(P)$ to a "semantically atomic" transition may not be restrictive.

**Proposition 2.10** *Suppose* $o \xrightarrow{l} o'$ *in an abstract derivative graph. Then for any* $P \in o$ *we have* $P \xrightarrow{\tau}{}^*_{\approx} \xrightarrow{l} P' \in o'$ *for some* $P'$. *Similarly if* $o \xrightarrow{\epsilon} o'$ *for any* $P \in o$ *we have* $P \xrightarrow{\tau}{}^*_{\approx} \xrightarrow{\tau}{}^{0,1} P'$.

**Proof:** We only show the first case. The second case is similar. Suppose not. Then, while for some $Q \in o$

$$Q \xrightarrow{\tau}{}^*_{\approx} \xrightarrow{l} Q' \in o'$$

for some $P \in o$, for no $P''$,

$$P \xrightarrow{\tau}{}^{*}_{\approx} P'' \xrightarrow{l} P' \in o'$$

Take $\{P_i\}_{i\in I}$ *s.t.* $P \xrightarrow{\tau}{}^{*}_{\approx} \xrightarrow{\tau}_{\not\approx} P_i \xrightarrow{l} P'$. But if no $P_i$ is bisimilar to $Q$, $P \not\approx Q$, contradiction, So $P_i \approx Q \approx P$, as required. ■

Now the graph $DG_{\approx}(P)$ can also be considered as a tree just by unfolding the derivatives, which is denote by $DG^{t}_{\approx}(P) = (N^L, E)$, assuming each node is labeled by equivalence classes on $[P]_{\approx}$ via a labelling function $L$.

The following is an important property of $DG_{\approx}(P)$, whose proof relies on the construction of Definition 2.9.

**Proposition 2.11** *For any $DG^{t}_{\approx}(P) = (N^L, E_{\approx})$, its automorphism (either respecting $L$ or not) is always identity.* ■

**Proof:** Since no two nodes in $DG_{\approx}(P)$ can have the isomorphic unfolding. If they do, then we can easily construct a bisimulation by which we are forced to adjoin these two nodes. Details are by induction on the depth of the first nodes rleted automorphism: See [6]. ■

**Definition 2.12** *A graph $G = (N^L, E)$ is isomorphic to another graph $G' = (N'^L, E')$ respecting e.g. labelling function $L$, written $(N^L, E) \simeq (N'^L, E)$, by usual graph theory.*

The following proposition shows that "semantically equivalent" programs have the same abstract graphs, and vice versa. By the result we are going to regard $DG_{\approx}(P)$ as a semantic basis of our cost theories.

**Proposition 2.13** $DG_{\approx}(P) \simeq DG_{\approx}(Q) \;\Leftrightarrow\; P \approx Q$

**Outline of Proof:** In the case of $\Rightarrow$, Since $DG_{\approx}(P)$ and $DG_{\approx}(Q)$ are isomorphic graphs, there exists a corresponding edge in $DG_{\approx}(Q)$ to the edge in $DG_{\approx}(P)$ and vice versa. Easily, $\{(P,Q) | DG_{\approx}(P) \simeq DG_{\approx}(Q)\}$ is a weak bisimulation. In the case $\Leftarrow$ is direct from Proposition 2.10. (note: $reach(P)_{WB} = reach(Q)_{\approx}$ if $P \approx Q$) ■

**Examples 2.14** (*Example of abstract derivative graphs*)
A part of abstract graph of Figure 1 is as follows:

$$[a.(\tau.P + \tau.\tau.P)]_{\approx} \stackrel{a}{\Longrightarrow} [P]_{\approx}$$

Now we about to define a cost graph. Before it, we define a pre-cost graph as any abstract derivative graph with cost data. A cost graph is defined as a pre-cost graph which correspond to a possible execution of a program, including branching structures.

**Definition 2.15** (*Pre-Cost graph*)
A pre-cost graph $(N^L, E, \rho)$ is an abstract derivative graph $DG_{\approx}(P) = (N^L, E)$ with a cost function $\rho : E \to Nat \cup \{\omega\}$.

**Notation 2.16**
An edge $e = o \stackrel{l}{\Longrightarrow} o'$ in a pre-cost graph, whose cost function is $\rho(e) = n$, is denoted by $o \stackrel{<l,n>}{\longrightarrow} o'$.

**Definition 2.17** (*Cost graph*)
A cost graph, denoted by $cg$,$cg'$, is a pre-cost graph $(N^L, E, \rho)$ for which there exists a map $\Psi : N \to \mathcal{P}$ such that:

1. $o \stackrel{<l,n>}{\longrightarrow} o' \;\Rightarrow\; \Psi(o) \stackrel{\tau}{\longrightarrow}_{\approx}^{n-1} \stackrel{l}{\longrightarrow} \Psi(o')$ where $\Psi(o) \in o^L$ and $\Psi(o') \in o'^L$.
2. $o \stackrel{<\epsilon,n>}{\longrightarrow} o' \Rightarrow \begin{cases} \Psi(o) \stackrel{\tau}{\longrightarrow}_{\approx}^{n-1} \stackrel{\tau}{\longrightarrow}_{\not\approx} \Psi(o') \text{ where } \Psi(o) \in o^L \text{ and } \Psi(o') \in o'^L \\ \Psi(o) \stackrel{\tau}{\longrightarrow}_{\approx}^{n} \Psi(o') \not\longrightarrow \text{ where } \Psi(o) \in o^L \text{ and } \Psi(o') \in o'^L \end{cases}$
   $(n \geq 1)$
3. $o \stackrel{<\epsilon,\omega>}{\longrightarrow} o' \;\Rightarrow\; \forall n\ \Psi(o) \stackrel{\tau}{\longrightarrow}_{\approx}^{n} \Psi(o')$ where $\Psi(o) \in o^L$ and $\Psi(o') \in o'^L$.
4. Moreover, if none of the above conditions hold for $L(o)$, we let $o \stackrel{<\epsilon,0>}{\longrightarrow} o'$.

Some discussions on Definition 2.17 are given.

1. By Proposition 2.10, 1 is well defined.
2. For 2 and 3, if $o \stackrel{\epsilon}{\longrightarrow} o'$ originally with $L(o) = L(o')$, a corresponding transition $P \stackrel{\tau}{\longrightarrow}_{\approx}^{n}$ with $n \neq 0$ can be:
   (a) $P'$ has further transition of $\stackrel{\tau}{\longrightarrow}_{\approx}^{*}$.
   (b) $P' \stackrel{l}{\longrightarrow} and/or P' \stackrel{\tau}{\longrightarrow}_{\not\approx}$, or $P' \not\longrightarrow$.

   If (a) is the only case, we keep adjoining $\stackrel{\tau}{\longrightarrow}_{\approx}$. If not (a), then we select (any of) the holding cases. If both, we can again select either. Whatever the case, by Proposition 2.10, for each original $o \stackrel{\alpha}{\longrightarrow} o'$, we have a corresponding $o \stackrel{<\alpha,n>}{\longrightarrow} o'$.
3. In the above scheme, if (a) continues to hold forever, then $o \stackrel{<\epsilon,\omega>}{\longrightarrow}$ results.
4. Note, if $o \stackrel{<\epsilon,0>}{\longrightarrow} o'$, necessarily $\Psi(o) \equiv \Psi(o')$. Conversely, suppose $o \stackrel{\epsilon}{\longrightarrow} o'$(originally) with $\Psi(o) \equiv \Psi(o)$. Given $\Psi(o)$, $\Psi(o) \stackrel{\tau}{\longrightarrow} P' \;\Rightarrow\; \Psi(o) \not\approx P'$ or $\neg\Psi(o) \stackrel{\tau}{\longrightarrow}$, none of the above labelling is possible, so we use $< \epsilon, 0 >$.
5. To summarize, the scheme counts $\stackrel{\tau}{\longrightarrow}_{\approx}$ steps a program needs for it to achieve any meaningful event(state change, non-$\tau$ action,termination). Note that state change should be considered to respect branching structures of underlying bisimilarity. If we take other equivalences, other ideas would be adapted. But the present one is most faithful to the underlying semantic structure, i.e. $DG_{\approx}(P)$.

**Examples 2.18** (*Examples of cost graph of programs*)

1. $a.\tau.0$
   Using the case 1 and second half of 2 of Definition 2.17,
   Since $a.\tau.0 \xrightarrow{a} \tau.0 \xrightarrow{\tau}{}^{1} 0$, we have $o \xrightarrow{<a,1>} o' \xrightarrow{<\epsilon,1>} o''$.
   Here $o^L = a.\tau.0$ and $o'^L = \tau.0$ and $o''^L = 0$.

2. $\tau.a + \tau.b$
   Using the case 1 and first half of 2 of Definition 2.17 since $\tau.a + \tau.b \not\approx a \not\approx b$,
   Since $\tau.a + \tau.b \xrightarrow{\tau} a \xrightarrow{a} 0$, we have $o \xrightarrow{<\epsilon,1>} o_a \xrightarrow{<a,1>} o'$.
   Since $\tau.a + \tau.b \xrightarrow{\tau} b \xrightarrow{b} 0$, we have $o \xrightarrow{<\epsilon,1>} o_b \xrightarrow{<b,1>} o'$.
   Here $o^L = \tau.a + \tau.b$ and $o_a^L = a$ and $o_b^L = b$ and $o'^L = 0$.

3. $\Omega \stackrel{\text{def}}{=} \tau.\Omega$
   Using the case 3 of Definition 2.17,
   Since $\forall n \; s.t. \; \Omega \xrightarrow{\tau}{}^{n} \Omega$, we have $o \xrightarrow{<\epsilon,\omega>} o$ where $o^L = \Omega$.

There may be several execution paths of a program $P$, since a concurrent system includes the branching structure whose components are "semantically" equal. The following is defined considering this property.

**Definition 2.19** (*Cost graphs of a program $P$*)
A cost graphs of $P$,written $\mathcal{CG}(P)$, is a set of all cost graphs for $P$.

**Examples 2.20** (*Examples of cost graphs*)
A part of a cost graph of Figure 1 is defined as follows:
a cost graph

$$a.(\tau.P + \tau.\tau.P) \stackrel{<a,2>}{\Longrightarrow} P$$

and a cost graph

$$a.(\tau.P + \tau.\tau.P) \stackrel{<a,3>}{\Longrightarrow} P$$

These cost graphs are shown in Figure 2.

The following definition and proposition shows that, for any cost graph of $P$, the cost graph is isomorphic to the abstracted derivative graph of $P$.

**Definition 2.21** (*Function from cg to $DG_{\approx}(P)$*)
Let $cg = (N^L, E) \in \mathcal{CG}(P)$ and $DG_{\approx}(P) = (N_{\approx}, E_{\approx})$. $\phi =< \phi_N, \phi_E >$ such that $\phi_N : N \to N_{\approx}, \phi_E : E \times \rho \to E_{\approx}$ is defined as follows:

$$\phi_N(o) = o$$
$$\phi_E(o \xrightarrow{<l,m>} o') = \phi_N(o) \stackrel{l}{\Longrightarrow} \phi_N(o')$$

By Proposition 2.13, we have the following essential result. We omit the easy proof.

**Proposition 2.22** *$\phi(cg) \simeq DG_{\approx}(P)$ iff $cg \in DG_{\approx}(Q)$ with $P \approx Q$*

This proposition allows us to compare between two programs which are semantically equal.

# 3 Various orderings on terms using cost graphs

In this section, we show several orderings on CCS-terms using cost graphs. Then we show the orderings between the sets of cost graphs using the order over cost graphs.

## 3.1 Comparison between two cost graphs

There are several ways to compare two cost graphs. This comparison is the basis to compare concurrent programs in terms of cost.

**Definition 3.1** (*Comparable graphs*)
Two cost graphs, say $cg$ and $cg'$, are comparable iff $\phi(cg) \simeq \phi(cg')$.

By proposition 2.22, $cg$ and $cg'$ are comparable iff they are bisimilar programs.

**Definition 3.2** (*Cost order on cost graph*)
Let $cg = (N^L, E, \rho)$ and $cg' = (N'^L, E', \rho')$ be two comparable cost graphs. Then $cg$ is faster than $cg'$, written $cg \leq_c cg'$, iff for a bijection $\Psi$ from $N^L$ to $N'^L$, we have, for some $n$ such that $n \geq m$,

$$\Psi(o) \stackrel{<\alpha,n>}{\longrightarrow} \Psi(o') \;\Leftrightarrow\; o \stackrel{<\alpha,m>}{\longrightarrow} o'$$

where $\geq$ is the usual order relation on natural numbers. ■

Note that by Proposition 2.11, when two comparable cost graphs are given, there is only one way to define such $\Psi$.

**Proposition 3.3** *$\leq_c$ is a preorder relation.*

**Proof:** Clearly, $cg \leq_c cg$. Moreover $cg \leq_c cg' \leq_c cg'' \;\Rightarrow\; cg \leq_c cg''$. ■

If a cost graph of one program is comparable to that of another program then they are bisimilar, by Proposition 2.22 and Definition 3.1 and 3.2.

**Proposition 3.4** $cg \leq_c cg' \wedge cg \in \mathcal{CG}_P \wedge cg \in \mathcal{CG}_Q \;\Rightarrow\; P \approx Q$ ■

## 3.2 Comparison between two sets of cost graphs

There are several ways to compare two sets of cost graphs. In concurrent programs, there are several possible execution paths. Thus, in general, *several* cost graphs are generated by *a* programs. Thus comparison between two sets of cost graphs means comparison between every possible execution paths of two programs. By the way of ordering the sets of cost graph, there are several ways of ordering two sets of cost graphs.

**Definition 3.5** (*Cost Preorders on Program*)
We define cost preorders on program as follows.

$$\begin{cases} 1 & P \sqsubseteq_{random} Q & \Leftrightarrow & \exists cg_p \in \mathcal{CG}(P).\ \exists cg_q \in \mathcal{CG}(Q).\ cg_p \leq cg_q \\ 2 & P \sqsubseteq_{ro} Q & \Leftrightarrow & \forall cg_q \in \mathcal{CG}(Q).\ \exists cg_p \in \mathcal{CG}(P).\ cg_p \leq cg_q \\ 3 & P \sqsubseteq_{rw} Q & \Leftrightarrow & \forall cg_p \in \mathcal{CG}(P).\ \exists cg_q \in \mathcal{CG}(Q).\ cg_p \leq cg_q \\ 4 & P \sqsubseteq_{r} Q & \Leftrightarrow & P \sqsubseteq_{r} Q \wedge P \sqsubseteq_{r'} Q \\ 5 & P \sqsubseteq_{o} Q & \Leftrightarrow & \exists cg_p \in \mathcal{CG}(P).\ \forall cg_q \in \mathcal{CG}(Q).\ cg_p \leq cg_q \\ 6 & P \sqsubseteq_{w} Q & \Leftrightarrow & \exists cg_q \in \mathcal{CG}(Q).\ \forall cg_p \in \mathcal{CG}(P).\ cg_p \leq cg_q \\ 7 & P \sqsubseteq_{ow} Q & \Leftrightarrow & P \sqsubseteq_{o} Q \wedge P \sqsubseteq_{w} Q \\ 8 & P \sqsubseteq_{so} Q & \Leftrightarrow & \forall cg_p \in \mathcal{CG}(P).\ \forall cg_q \in \mathcal{CG}(Q).\ cg_p \leq cg_q \end{cases}$$

$\sqsubseteq_{random}$ is called *random cost preorder.* $\sqsubseteq_{rw}$ is called *relative optimal cost preorder.* $\sqsubseteq_{ro}$ is called *relative worst cost preorder.* $\sqsubseteq_{r}$ is called *relative cost preorder.* $\sqsubseteq_{o}$ is called *optimal cost order.* $\sqsubseteq_{w}$ is called *worst cost order.* $\sqsubseteq_{ow}$ is called *optimal-worst cost order.* $\sqsubseteq_{so}$ is called *strong optimal cost order.* ■

**Discussion 3.6**

1. $P \sqsubseteq_{random} Q$ shows that If $P$ and $Q$ run, the run of $P$ may be faster than the run of $Q$.
2. $P \sqsubseteq_{ro} Q$ means in comparison with any run of $Q$, that $P$ may run faster. So this is a *speculative* order.
3. $P \sqsubseteq_{rw} Q$ is important: This says that $P$ is more stable than $Q$, in the sense that, if we let $P$ run, it is always possible that $Q$ does worse than that.
4. $P \sqsubseteq_{r} Q$ is a sound way of combing previous two.
5. Sometimes we require that, if P does its (one of) the best, then it always beats $Q$: this is $P \sqsubseteq_{o} Q$, measuring $P$'s efficiency at its best, strengthening $\sqsubseteq_{ro}$.
6. $P \sqsubseteq_{w} Q$ compares $P$ and $Q$ at $Q$'s (one of the) worst. This is clearly the strengthening of $\sqsubseteq_{rw}$.
7. $P \sqsubseteq_{ow} Q$ is a sound way of combing the previous two.
8. $P \sqsubseteq_{so} Q$ assures that at any possible run, $P$ is always faster. In other words, the worst runs of $P$ is better than the best runs of $Q$.

We show the examples of various ordering

**Examples 3.7** (*Examples of comparison of cost graph*)
For $a + \tau.\tau.a$, there are two cost graphs $o \xrightarrow{<a,1>} o'$ and $o \xrightarrow{<a,3>} o'$. For $\tau.a$, there is a cost graph $o \xrightarrow{<a,2>} o$.

1. $a \sqsubseteq_{so} \tau.a$ and $a \sqsubseteq_{so} a + \tau.\tau.a$ But, $\tau.a \not\sqsubseteq_{so} a + \tau.\tau.a$.
   Their means $a$ is always faster than $\tau.a$ and $a + \tau.\tau.a$. But, $\tau.a$ is not always faster than $a+\tau.\tau.a$. Precisely speaking, $\tau.a$ is reduced to 0 at two steps($\tau$ and $a$). $a+\tau.\tau.a$ takes at least one step, while at most three steps.

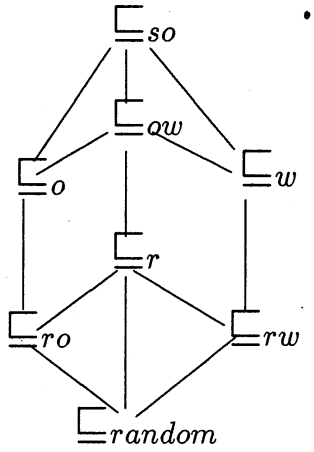


Figure 3: Inclusion relation on cost orders

2. $\tau.a \sqsubseteq_w a + \tau.\tau.a$. But, $\tau.a \not\sqsubseteq_o a + \tau.\tau.a$
   There exists a run $\tau.\tau.a$ is slower than a run $\tau.a$ of a term $\tau.a$. So $\tau.a$ is stable w.r.t. cost. But, This also means that $\tau.a$ is not faster than every run(path) of $a + \tau.\tau.a$. That is $\tau.a$ is not always faster than $a + \tau.\tau.a$. So, $\tau.a$ is not optimal w.r.t. cost.

3. $a + \tau.\tau.a \sqsubseteq_o \tau.a$. But, $a + \tau.\tau.a \not\sqsubseteq_w \tau.a$
   This example is reverse of the previous example. $a + \tau.\tau.a$ contains the optimal run $a$. But $\tau.a$ not contain the worst path.

4. $a + \tau.\tau.a \not\sqsubseteq_{so} a + \tau.\tau.a$
   It means that $\sqsubseteq_{so}$ is not a preorder since it is not reflexive.

The following shows inclusion among these preorders.

**Proposition 3.8** (*Inclusion relation among cost orders*)

1. $\sqsubseteq_{so} \subseteq \sqsubseteq_o \subseteq \sqsubseteq_{ro} \subseteq \sqsubseteq_{random}$
2. $\sqsubseteq_{so} \subseteq \sqsubseteq_w \subseteq \sqsubseteq_{rw} \subseteq \sqsubseteq_{random}$

■

**Proof:** 1 and 2 are direct from definitions of each preorder. ■

The inclusion results are shown in Figure 3. Other than the transitive closures of the inclusion, no comparison is possible. Moreover all depicted inclusions are proper at least for CCS terms.

# 4 Relationship between orders of cost graphs and efficiency preorder

## 4.1 Efficiency preorder

In Introduction, we discussed the efficiency preorder for CCS, which was proposed in [1]. This preorder is based on the number of $\tau$ transitions. The preorder is defined as follows[2]:

**Definition 4.1** (*Orders on label*)
Let $\leq$ be the binary relation on $Act^*$ generated by inequations $s \leq \tau s$ and $s \leq s$ i.e. $\leq$ is closed under reflexivity, transitivity, and substitution under catenation contexts. ■

**Definition 4.2** (*Efficiency Prebisimulation*)
A binary relation $R \subseteq \mathcal{P} \times \mathcal{P}$ is an efficiency prebisimulation if $\forall (P,Q) \in R$ and

1. $P \xrightarrow{s} P' \Rightarrow \exists s'\ s \leq s'\ \exists Q'\ Q \xrightarrow{s'} Q' \wedge (P',Q') \in R$
2. $Q \xrightarrow{s'} Q' \Rightarrow \exists s\ s \leq s'\ \exists P'\ P \xrightarrow{s} P' \wedge (P',Q') \in R$ ■

**Definition 4.3** (*Efficiency preorder* $\sqsubseteq_E$)

1. $\sqsubseteq_E = \bigcup\{R | R \text{ is a efficient prebisimulation}\}$
2. $P \sqsubseteq_E Q$ iff $\exists$ efficiency prebisimulation $R, P\mathbf{R}Q$ ■

**Lemma 4.4** $P \sqsubseteq_E Q \Rightarrow P \approx Q$

## 4.2 Correspondence between cost preorder and efficiency preorder

In this section, we show the correspondence between cost preorder presented in this paper and efficiency preorder. Firstly, we show the following lemma that cost preorder can be defined inductively. Then, we show that cost relative order is indeed an efficiency bisimulation.

**Lemma 4.5** The subtree of a cost graph is a cost graph.

**Lemma 4.6**

$$P \sqsubseteq_r Q \text{ iff } \begin{cases} P \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{\alpha} P' \Rightarrow \exists n\ s.t. m \leq n \wedge Q \xrightarrow{\tau}{}^{n}_{\approx} \xrightarrow{\alpha} Q' \wedge P' \sqsubseteq_r Q' \\ Q \xrightarrow{\tau}{}^{n}_{\approx} \xrightarrow{\alpha} Q' \Rightarrow \exists m\ s.t. m \leq n \wedge P \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{\alpha} P' \wedge P' \sqsubseteq_r Q' \end{cases}$$ ■

**Proof:**
($\Rightarrow$): We should prove three cases of $\xrightarrow{\alpha}$. Since three cases are similar, we show one of them.

[2]Our definition reverses left and right for original definition

1. $P \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{l} P'$. By Definition 2.17,

$$P \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{l} P' \;\Rightarrow\; o \overset{<l,m+1>}{\longrightarrow} o' \wedge o^L = P \wedge o'^L = P'$$

By Assumption for any $cg \in \mathcal{CG}(P)$, there exists $cg' \in \mathcal{CG}(Q)$ such that $cg \leq cg'$. Let $cg = (N, E, \rho)$ and $cg' = (N', E', \rho')$. Then by Definition 3.1, there exists a bijection $\Psi : N^L \to N'^L$ such that $m \leq n$ and

$$o \overset{<l,m+1>}{\longrightarrow} o' \;\Leftrightarrow\; \Psi(o) \overset{<l,n+1>}{\longrightarrow} \Psi(o')$$

If $\Psi(o) \overset{<l,n+1>}{\longrightarrow} \Psi(o')$, then $P \xrightarrow{\tau}{}^{n}_{\approx} \xrightarrow{l} P'$ where $\Psi(o)^L = Q$ and $\Psi(o)^L = P'$. While if $cg \leq cg'$, then for any $cg'' \in \mathcal{CG}(P')$, there exists $cg''' \in \mathcal{CG}(Q')$ which is a subtree of $cg' \in \mathcal{CG}(Q)$ such that $cg'' \leq cg'''$. Hence $P' \sqsubseteq_r Q'$.

2. $P \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{\tau}_{\not\approx} P'$,

3. $P \xrightarrow{\tau}{}^{m+1}_{\approx} P'$. In both cases, similarly to 1 using the edge $\overset{<\epsilon,n>}{\longrightarrow}$.

($\Leftarrow$): We should prove that for any $cg \in \mathcal{CG}(P)$ there exists $cg' \in \mathcal{CG}(Q)$ such that $cg \leq cg'$. There are three type of the edge.

1. Case $o \overset{<l,n>}{\longrightarrow} o'$:

$$\begin{array}{lll}
o \overset{<l,n>}{\longrightarrow} o' & \Rightarrow & P \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{l} P' \wedge o^L = P \wedge o'^L = Q \quad \text{(Definition 2.17)} \\
& \Rightarrow & \exists n.(m \leq n) \wedge Q \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{l} Q' \wedge P' \sqsubseteq_r Q' \quad \text{(Assumption)} \\
& \Rightarrow & o'' \overset{<l,n>}{\longrightarrow} o''' \wedge o''^L = Q \wedge o'''^L = Q' \wedge P' \sqsubseteq_r Q' \quad \text{(Definition 2.17)} \\
& \Rightarrow & o'' \overset{<l,n>}{\longrightarrow} o''' \wedge o''^L = Q \wedge o'''^L = Q' \\
& & \wedge \forall cg'' \in \mathcal{CG}(P') \exists cg''' \in \mathcal{CG}(Q) s.t. cg'' \leq cg''' \quad \text{(Definition 3.5)}
\end{array}$$

Then for any $cg$ containing $cg''$, there exists $cg' \in \mathcal{CG}(Q)$ containing $cg'''$.

2. Case $o \overset{<\epsilon,n>}{\longrightarrow} o'$: Therefore, we divide two cases. That is $P \xrightarrow{\tau}{}^{m}_{\approx} \xrightarrow{\tau}_{\not\approx} Q$ and $P \xrightarrow{\tau}{}^{m+1}_{\approx} Q$. In both cases, we can prove similarly to 1. ■

**Theorem 4.7** $P \sqsubseteq_r Q \;\Rightarrow\; P \sqsubseteq_E Q$ ■

**Proof:** Assumes that there is a transition $P \xrightarrow{s} P'$. This transition can be divided into

$$P \xrightarrow{\tau}{}^{m^0}_{\approx} \xrightarrow{\tau}_{\not\approx} P_1^1 \xrightarrow{\tau}{}^{m_1^1}_{\approx} \xrightarrow{\tau}_{\not\approx} \cdots P_1^j \xrightarrow{\tau}{}^{m_j^1}_{\approx} \xrightarrow{l_1} P_1 \xrightarrow{\tau}{}^{m_1^2}_{\approx} \xrightarrow{\tau}_{\not\approx} \cdots P_i^j \xrightarrow{\tau}{}^{m_j^i}_{\approx} P'$$

Then, by Lemma 4.6,

$$\begin{array}{llll}
P \sqsubseteq_r Q & then & P \xrightarrow{\tau}{}^{m^0}_{\approx} \xrightarrow{\tau}_{\not\approx} P_1^1 \;\Rightarrow\; \exists n^0\, s.t. m^0 \leq n^0 \wedge Q \xrightarrow{\tau}{}^{n^0}_{\approx} \xrightarrow{\tau}_{\not\approx} Q_1^1 \wedge P_1^1 \sqsubseteq_r Q_1^1 \\
P_1^1 \sqsubseteq_r Q_1^1 & then & P_1^1 \xrightarrow{\tau}{}^{m_1^1}_{\approx} \xrightarrow{\tau}_{\not\approx} P_1^2 \;\Rightarrow\; \exists n_1^1\, s.t. m_1^1 \leq n_1^1 \wedge Q_1^1 \xrightarrow{\tau}{}^{n_1^1}_{\approx} \xrightarrow{\tau}_{\not\approx} Q' \wedge P_1^2 \sqsubseteq_r Q_1^2 \\
\vdots & & \vdots \\
P_i^j \sqsubseteq_r Q_i^j & then & P_i^j \xrightarrow{\tau}{}^{m_j^i}_{\approx} P' \;\Rightarrow\; \exists n_j^i\, s.t. m_j^i \leq n_j^i \wedge Q_i^j \xrightarrow{\tau}{}^{n_j^i}_{\approx} Q' \wedge P' \sqsubseteq_r Q'
\end{array}$$

Then, $P \sqsubseteq_r Q \;\Rightarrow\; P \xrightarrow{s} P' \;\Rightarrow\; \exists s'\, s \leq s' \exists Q'\, Q \xrightarrow{s'} Q' \wedge P' \sqsubseteq_r Q'$.

By Definition 4.2, this shows $\sqsubseteq_r$ is an efficiency prebisimulation. ■

We do not know whether the converse inclusion holds or not.

# 5 Conclusion

In this section we compare our construction with Arun-Kumar-discuss some further issues.

## Comparison with efficiency preorder

There are several differences between our work and Arun-Kumar-Hennessy's construction. As was discussed already, our cost graphs are intended to provide a comprehensive framework for evaluating and analysing concurrent programs in regard of their efficiency. Specifically varied ways of measuring efficiency of programs according to the need are a feature not found in the precursor. Especially the relative optimal and strongly optimal orderings seem to have practical significance. The second difference is the semantic framework. Efficiency preorder is based on inductive definition, which would make the reasoning easier for some cases. However we believe, especially in the case when the reasoning can be performed semi-algorithmically, both methods may not differ so much in practice. Finally we should note that two frameworks are different in the ways of counting cost of two different transitions which do the same thing but whose "intermediate semantic points" may be different (this is counted in $\sqsubseteq_E$ but not in ours). Indeed it can be proved that, if we relax this point, $\sqsubseteq_r$ easily coincides with $\sqsubseteq_E$ which shows how our construction of $\sqsubseteq_r$ is at least close to $\sqsubseteq_E$. While this and other options are possible, our intention is to start from the formulation which is most faithful to the underlying semantic structure, i.e. abstract derivation graphs. In practice various ramifications should be considered.

## Issues

There are several problems which are to be solved for the application of the present theory. The first thing, maybe the most difficult, is how the present framework can be mapped to the *real* cost of execution environments. In this context, cost theory should also include the soundness of mapping between programs and their execution, w.r.t. cost orderings. But the significance of the present construction also lies in that, at least in the stipulated criteria, ways of ordering programs, either their execution images or source programs, are now present.

As a theoretical basis for pragmatic development, algorithmic aspects of cost theory should be performed. Apart from such issues as algorithmic construction of cost graphs for e.g. finite-state programs, not only comparison of programs but also optimization methodologies can be developed, once we are given a firm criteria about efficiency. We hope that many optimization technologies in sequential programs will be adaptable in concurrency settings, but very probably a special method may be needed for concurrent systems. This point is also related with the optimal execution graph of a given program, as well as the existence of optimal programs among weakly bisimilar ones.

In conclusion, in this paper we have investigated the possibility of capturing the notion of cost in concurrency (specifically the one of the execution time) in a rigorous formal framework. While many things remain to be solved, we hope that our work provides a possible formal foundation for measuring efficiency of concurrent programs.

## Acknowledgments

The author thank Kohei Honda for discussions and comments on the paper and proof-reading.

## References

[1] S. Arun-Kumar and M. Hennessy. *An efficiency preorder for processes.* Acta Informatica 29, pages 737–760, Springer-Verlag, 1992.

[2] H. Barendregt. *The Lambda Calculus:Its syntax and semantics.* North Holland, revised edition edition, 1984.

[3] J. Hindley and J. Seldin. *Introduction to Combinators and $\lambda$ calculus.* Cambrige University Press, 1986.

[4] T. Hoare. *Comunicating Sequential Processes.* 1978.

[5] K. Honda and M. Tokoro. *An Object Calculus for Asynchronous Communication. ECOOP'91*, LNCS 512, pp.21–52, Springer-Verlag 1991.

[6] M. Kubo. *Notes on unfolding cost graph*, manuscript.

[7] J-L. Levy. *Optimal reduction in th lambda-calculus.* In J-P. Seldin and J-R. Hindley, editors, To H.B.Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, pages pp159–191, Academic Press, 1980.

[8] R. Milner. *A Calculus of Communicating Systems. LNCS 92*, Springer-Verlag, 1980.

[9] R. Milner. *Communication and Concurrency.* Prentice Hall, 1989.

[10] R. Milner. *Function as Processes. Mathematical Structure in Computer Science*, 2(2), pp.119–146, 1992.

[11] R. Milner, J. Parrow, and D. Walker. *A calculus of Mobile Processes PartI/II.* Technical Report ECS-LFCS-89-85/86, Laboratory for Foundations of Computer Science, University of Edinburgh, June 1989.